The Cluster Integration Toolkit

An Extensible, Portable, Scalable Cluster Management Software Implementation

James H. Laros III[1]*, Lee Ward[1], Nathan W. Dauchy[2], James Vasak[2], Ruth Klundt[3], Glen Laguna[1], Marcus Epperson[1], Jon R. Stearley[1]

Sandia National Labs. USA

Abstract. This paper describes the implementation of a software architecture that was described in *"An Extensible, Portable, Scalable Cluster Management Software Architecture"*[4]. This implementation, named Cluster Integration Toolkit (CIToolkit, or just CIT), has been used to successfully integrate and support numerous cluster systems in production at Sandia National Labs and other sites, the largest of which is 1861 nodes. This paper discusses the goals of the CIToolkit and how our implementation achieved the stated goals. We describe the installation process and how tasks common to cluster implementation and support are automated. A set of tools used to maintain the system both during and after installation are also discussed.

1 Introduction

Cluster integration and management is a very broad topic. To be complete, a set of integration tools must be useful in every step of cluster integration from initial hardware configuration and testing to final production support of the resulting cluster. Frequently, approaches to cluster integration and management focus on specific aspects of this overall task or target specific hardware or configurations of that hardware. We believe that to achieve long-term benefit and use from a cluster integration and management toolkit, that toolkit must exploit the similarities of all clusters rather than focus on the specifics of individual installations. To be useful on a wide range of clusters comprised of different hardware and connected in various topologies, a cluster toolkit must be extensible (capable of accommodating both existing and future hardware components easily) and portable (the same software package can be used to support these different clusters presenting the same interface to the user). This same toolkit must provide these capabilities with the ability to scale to thousands of nodes to be useful on large cluster systems. These are steep requirements, but achievable as we will demonstrate.

The CIToolkit is a set of software tools for configuring, testing, and managing Linux^{TM}[5] clusters. The CIToolkit is designed to be architecture and hardware independent and provide a stable platform for all run-time environments and

* Corresponding Author: James H. Laros III, jhlaros@sandia.gov

user applications. The goals of the project are described in Section 3 (Goals). Specifics on the implementation of these goals, the software architecture and how it was implemented are detailed in Section 4 (Implementation). The process outlining the general steps of a cluster install is discussed in Section 5 (Installation Process). The layered nature of the toolkit and how it accommodates the support of a wide range of capabilities along with a discussion of some of the existing capabilities is included in Section 6 (Tools). Throughout the paper we relate the utility of the CIToolkit to our experiences in cluster integration and maintenance.

2 Related Work

Many approaches have been taken in the development of management tools for clusters. Few have addressed support of diskless nodes, and those that have do not adequately address scalability. Frequently, management packages address only a few of the issues involved in the complete integration of a cluster. These solutions are often cluster and/or topology specific and are difficult to utilize or port when cluster configurations change. Our efforts to address these issues began in 1997, and, while very few organized efforts existed at that time, much progress has been made in addressing many of the issues related to cluster integration and management. Unfortunately no single effort, even now, addresses all of the functional and performance requirements that we set out to meet in 1997. Please reference *"An Extensible, Portable, Scalable Cluster Management Software Architecture"*, for our survey of related efforts and how they contrast with our requirements.

3 Goals

While the CIToolkit effort was started to support the Cplant$^{\mathrm{TM}}$[6] run-time system, the decision was made to broaden the scope of the effort. Our own experiences with cluster installations emphasized the importance of taking a generic approach which would incorporate as many possible scenarios as we could envision. We extended this goal in an attempt to account for hardware, topologies and cluster applications that we have not yet envisioned. The emergent goal was to make as few assumptions as possible in the design and subsequent implementation of the CIToolkit. The toolkit's configuration database generation process illustrates this well. It allows the user to represent their particular cluster in the database – the devices they have used and how they are connected – without imposing our notion of how clusters should be architected. The net result of this is that the CIToolkit should work on most any cluster existing today and still be useful in the future as cluster designs continue to evolve.

Automating as much of the integration process as possible is another goal. The discovery process, for example, automates the population of the database with hardware specific information and optionally configures devices, expediting initial cluster integration. Diagnostic tools, which can leverage the database to understand cluster hardware and topology, can simplify hardware testing and troubleshooting tasks, especially during the initial burn-in phase of cluster integration when infant mortality rates are high. For diskless clusters, generation of the "bootable hierarchy" is automated based on the cluster's database. Other configuration file generation programs also ease the task of cluster configuration.

Ease of cluster management after integration is complete is another very important goal. Our aim is to supply a cluster management interface to the user that does not require "cluster-experts" to understand and use. Our target cluster administrator only needs general UNIX knowledge and a basic understanding of the cluster to be administered. This interface should allow the administration of a cluster as a single system regardless of how large the cluster is. Tools are also provided for more advanced users to allow them to get the most out of the cluster hardware, but are in no way required for basic operation. All this makes the process of bringing up a cluster more accessible to novice users and far less error prone for even the most experienced integrators.

Another goal that guided our design and implementation is support for diskless nodes. Diskless nodes are an important consideration for sites that have the requirement of switching between classified and unclassified environments. We have found that diskless booting can ease management and upgrade tasks as well as reduce the number of moving parts in the cluster, which can increase overall system reliability. Unfortunately, supporting diskless clusters that scale to thousands of nodes is an area that has not been given much attention outside our efforts. A hardware architecture that is structured hierarchically can be very useful in improving the scalability of diskless clusters. Booting, command execution, and monitoring are a few areas where a hierarchically designed system can enhance scalability and performance. This requires our toolkit to inherently support flat or hierarchical hardware topologies, which is not a common capability among cluster management suites.

Due to the overhead and possible complexity involved with supporting local modifications for new releases, we have a stated goal of avoiding kernel modification. We have successfully enforced this in our implementation in a way that doesn't prevent support of other efforts that have employed this approach. For instance, we could leverage the work that has been done with LinuxBios[7] without affecting the remainder of the toolkit.

Our target run-time system (Cplant) imposes the requirement that we cannot affect the performance of compute nodes with our management software. This generally means that we do not require any daemon processes on compute nodes to accomplish monitoring and statusing, and otherwise managing the nodes.

To accomplish our goal of supporting virtually any run-time software, we have chosen to cleanly separate cluster management and run-time environments. This allows the CIToolkit to support the widest possible range of cluster uses. Database clusters, web farms, and other types of clusters can be managed just as easily with the CIToolkit as high performance compute clusters for scientific applications. We have also employed run-time support for concepts like virtual machines that provide soft partitioning of a cluster to as many logical clusters as the user desires. This concept allows different configurations, versions, or completely different run-time systems to share the same physical cluster. We maintain our single management interface even when this capability is exercised.

4 Implementation

4.1 Design

The software architecture that the CIToolkit is based on is described in detail in *"An Extensible, Portable, Scalable Cluster Management Software Architecture"*. In this section we briefly present some of the concepts covered in the afore-mentioned paper that are used throughout our implementation discussion. In brief, the architecture supports integration and management of the widest possible range of clusters using common foundational components: Device Class Hierarchy, Persistent Object Store, and Layered Utilities.

The Device Class Hierarchy provides a hierarchical organization of devices and their capabilities. This Device Class Hierarchy has been implemented in Perl[8] in an object-oriented methodology, which allows the toolkit to support existing devices of a wide range of type and functionality.

Inheritance allows us to leverage common characteristics and capabilities among general categories of devices. The hierarchical structure allows us to not only leverage this commonality when possible but also differentiate specific device characteristics and capabilities when necessary. Ultimately, each physical device type in the system will be encapsulated in a separate device class. These characteristics allow the Device Class Hierarchy to be easily extensible when support for new device types is required. These properties also provide higher-level tools with a consistent interface, regardless of the specific hardware used. Once described, these classes can be instantiated as necessary to construct a cluster description unique to that installation.

The Persistent Object Store serves as the basis of information for the cluster tools themselves. This configuration database, as it is generically referred to, is a representation of the physical cluster architecture; the devices of which it is comprised and how those devices are connected together, among other information. The devices that populate the database are instantiated objects derived from the Device Class Hierarchy. This methodology allows for the CIToolkit to support clusters comprised of any devices connected in any way. To further maintain flexibility, we do not require that the Persistent Object Store be implemented in any particular database. For its ease of use and lightweight implementation, GDBM[9] has been used at the majority of CIToolkit installations. However, through an API layer that is transparent to the tools in the toolkit, LDAP[10] or one of many Structured Query Language (SQL) based Database Management Systems (DBMS) may also be used.

Layered Utilities is the concept of building capabilities in functional strata starting with very specific, low-level tools and functions at the foundation and adding layers of increasingly higher-level, more intelligent and user-friendly interfaces. Examples of some of the resulting tools will be described in Section 6 (Tools).

4.2 Language

Perl was chosen as the language with which to implement the CIToolkit for many reasons. Perl provides the portability desired since it has been ported to virtually all UNIX system derivatives. By using Perl, components of the toolkit can be supported on any Linux distribution, flavor of UNIX, and even on a Microsoft^{TM}[11] platform, if required. Even embedded Linux systems have become a potential platform for portions of the toolkit. Recently, the CIToolkit was successfully ported to a Sharp Zaurus^{TM}[12] for the purposes of device configuration. Devices like the Zaurus could easily play a bigger role in cluster management. We expect embedded systems to take on increasing importance in clusters in the near future.

Perl was also chosen because portions of the software architecture mapped well into an object-oriented design paradigm. Perl supports this paradigm without forcing its use everywhere. Different portions of the implementation map to different programming methods all co-exist well in a Perl implementation. Perl also has the advantage of being a mature language. Many contributions to the core language have been made available through the Comprehensive Perl Archive Network (CPAN)[13]. In implementing the toolkit we have tried to incorporate any open-source effort that could be leveraged. CPAN modules have saved us massive amounts of time when they could be appropriately leveraged. Some of the CPAN modules that have been important to the development of the CIToolkit are:

- `Getopt::Declare`[13]
- `Net::Telnet`[13]
- `Net::Server`[13]

The `Getopt::Declare` module has saved us countless hours in writing our own command line parsing code. Many of the high-level tools support a large number of options that can be used in complicated combinations. The `Getopt::Declare` module has made this task much simpler, and has the added feature of automatically generating usage documentation. The `Net::Telnet` module has allowed for easy command and response sequences to be coded in the classes of the Device Class Hierarchy. This module takes care of the messy line protocol of telnet, making the methods in our device classes easy to develop, maintain and debug. The `Net::Server` package of modules enabled rapid creation of stable daemon processes for command distribution, monitoring and logging. They provide an easy to use abstraction for socket communications and offer many "personalities" as termed by the author, which can be used for different client server requirements.

Perl has also proved to be a comfortable language to work in, contributing to quick and clean implementations of our design. The implementation is not entirely Perl. Most notably are some of the Myrinet[14] diagnostics as well as third party open source test and benchmark codes that are included with the CIToolkit distribution and are, for the most part, written in C.

4.3 Modules

The implementation of the CIToolkit is modular. While there are some dependencies, the user can largely pick and choose modules depending on which portions of the toolkit are appropriate for their installation. The following modules are included in the full distribution:

- *base* - core of distribution
- *site* - separate "site-specific" commands/utilities/libraries
- *config* - configuration files and utilities
- *diskless* - support for diskless nodes
- *diskfull* - support for diskfull configurations
- *diag* - diagnostic commands/utilities/libraries
- *devtools* - various development assistance tools
- *myrinet* - drivers, support, and diagnostic tools specific to the myrinet interconnect
- *chits* - hardware issue tracking web/database interface
- *test* - test code for any of the above modules

Other modules and placeholders are present at our local site. For instance, the toolkit was developed to support the Cplant run-time effort, so a module called *cplant* is included and provides an example of how any run-time can be integrated into the toolkit with ease. Additional modules that could be added in the future include support for additional high speed interconnects or special hardware, advanced logging or monitoring, or a new run-time environment which would benefit from leveraging the CIToolkit tools and configuration information.

As the core of the toolkit, the *base* module contains:

- Libraries that define devices and their capabilities (device class hierarchy)
- Libraries that provide utilities to higher-level libraries and commands
- Database interface layer
- Layered utilities (Tools)

These broad categories indicate the types of things contained in the *base* module. Notable features of the *base* module, as well as some examples of other modules, are mentioned in more detail in Section 5 (Installation Process) and Section 6 (Tools).

5 Installation Process

The installation process of a cluster using our toolkit basically follows these steps.

- Initial Installation of CIToolkit Software
- Database (Persistent Object Store) Configuration
- Hardware Installation
- Cluster Software Customization and Configuration File Generation
- Initialization and Discovery
- Trouble-shooting and Diagnostics
- Installation of the Run-time Environment

The order of the first few steps is somewhat flexible, and can change depending on how well defined the cluster design is, when administrative node hardware is available, and how soon the integrators wish to begin leveraging CIToolkit capabilities. Even before the cluster hardware has been installed, tasks like assigning IP addresses, printing out device labels, and eventually generating wiring diagrams can be automated using the cluster database as an information resource. Another tool provided by the CIToolkit that can be useful early on in the process is the Cluster Hardware Issue Tracking System (CHITS). This web-based tool leverages the CIToolkit infrastructure and provides a tracking mechanism for hardware issues. CHITS is discussed in more detail later in this paper.

5.1 Initial Installation of CIToolkit Software

Installing the tools can be done at any time, even before the entire cluster is physically installed. The CIToolkit software is installed on what we call the administration node. However, if the admin node is not yet available, the base of the CIToolkit can be installed on an available workstation to facilitate configuration database creation and allow the integrator to leverage the capabilities mentioned above.

The installation process of the CIToolkit is automated with the "`make`" utility. At this point, the decision can be made about which modules to install and use. Additional modules can be installed later, if desired. The process is very simple and only requires a few edits to the top-level `Makefile`. The majority of the toolkit installs very quickly as it is simply copied into place because it is written in Perl and does not require compilation. Installing some of the modules, like the *myrinet* and *diag* modules, will compile the third party drivers and programs for you based on your edits in the `Makefile` and take just a bit longer to install. Other options that appear in the top-level `Makefile` are used to generate configuration files used by the CIToolkit for purposes like defining paths of executables and type of database used. The modules installed at a particular site can vary based on local site requirements and the composition of the cluster.

5.2 Database (Persistent Object Store) Configuration

Next the database, or more correctly, persistent object store, must be generated. This will be a database representation of your physical cluster, hardware and topology. As previously mentioned, these steps can be taken out of order when desired. The database generation process can be useful in architecting the cluster even before purchase. Topology details can be worked out when generating a detailed description of the cluster, which has the added benefit of potentially identifying aspects of system architecture or support devices that may have been overlooked.

There are a growing number of ways to accomplish generation of the database. Below, we describe both the configuration program method, which provides the most power and flexibility, and the command line method, which is a bit more approachable. The task of writing a configuration program is not hard for an experienced programmer with knowledge of simple data-structures and Perl syntax. Many example configuration programs are provided with the distribution ranging from simple programs that generate example cluster databases to programs that we have used to generate the databases for our largest systems. In the configuration program, the author instantiates objects of the classes that represent the devices that their particular cluster is made of. As part of this instantiation, the author links the objects together by attribute assignment to describe the network topology. This is conceptually straightforward but has been the point of the most criticism, since the process can be somewhat tedious and the programming can be complicated.

Novice programmers' difficulty with writing a configuration program led us to provide more approachable methods to generating the database. Recently, a command line interface for database generation has been added. While this interface is not infinitely flexible, as is the configuration program method, it should suffice for most cluster installations. This command line interface provides a series of commands to create new objects, assign attributes to the objects, and group the objects, or other groups, as desired. A simple flat-file populated with appropriate commands can easily serve as a database generation script. These command line tools are also very handy for editing the objects in the database after database generation for purposes like changing the IP address of an interface. After the database is generated various commands are available to view the entire database, objects or individual attributes. Database commands and interfaces are discussed further in Section 6 (Tools).

5.3 Hardware Installation

While the hardware installation process is mostly a manual task, some aspects of configuration can be aided by software tools. Configuration of some top-level devices such as external power controllers or terminal servers may require manual assignment of information - like IP address. As previously mentioned, the toolkit has been ported to a hand-held embedded Linux device, which could be used for this purpose. Since it has the same database which contains information about all of the devices in the cluster, the setup of these top-level devices becomes easier than wheeling a dumb terminal around to do the job.

5.4 Cluster Software Customization and Configuration File Generation

If a diskless cluster is being installed, the next step is generation of the bootable hierarchy. While the toolkit does not currently provide a mechanism for image cloning used in diskfull cluster installations, much work has been done in this area by others and any of the tools written to accomplish this task can be easily integrated into the CIToolkit. It is our belief that the bootable hierarchy concept can enhance the image cloning procedure by allowing for per-node differences in the distributed software, something that is not easily provided by current approaches. Generation of the bootable hierarchy is accomplished directly and completely from the information contained in the database. This process generates nfs-root, kernel, and configuration information on a per-node basis. This process is described in more detail in, "Implementing Scalable Diskless Cluster Systems using the Network File System (NFS)"[15]. This automates what would normally be a number of very tedious tasks. After initial generation, individual configuration files can be updated with the use of simple utilities provided for this purpose. An open-source utility cfengine[16] has been very valuable in this process. Our install process generates configuration files that are leveraged by the cfengine utility during installation.

Configuration files for both diskfull and diskless installations like `/etc/hosts`, and files which may only be necessary for diskless installations like `/etc/dhcpd.conf` are also automatically generated based on the cluster database using tools provided in the CIToolkit. Tools for the generation of many other configuration files which apply to diskless and diskfull installations are provided. Other tools are easy to develop by leveraging the cluster database, the Device Class Hierarchy, and other libraries included in the CIToolkit.

5.5 Initialization and Discovery

The discovery process does just what its name indicates: it discovers the system. The database contains a description of the devices that make up the cluster and topology information about how they are connected. For the purposes of the discovery process, this information tells the discover utility how to get to the device being discovered. The discovery process uses this information to connect to the device and obtain specific information, such as a MAC address, and store it in the database. This information is important for generating dhcp configuration files, for example. The discovery process also optionally configures the device to operate appropriately as part of the cluster. Site-specific configuration programs can be written to configure devices based on local requirements. Several examples of these are contained in the *site* module.

5.6 Troubleshooting and Diagnostics

Although presented here as part of the installation process, the diagnostic tools of the CIToolkit are useful during every phase of cluster life, from installation to maintenance and troubleshooting. The tools include both locally written and open-source utilities that range in applicability from all devices in a cluster to specific hardware, such as Myrinet. As mentioned, we exploit open-source as much as possible and readily include tools we have found useful with the distribution.

The *diag* module tests and benchmarks can be broken down into two major categories: single node tests and full system/interconnect tests. The single node tests include common open source benchmarks like Stream[17], Memtest[18], Linpack[19], hdparm[20] and the NAS[21] kernels. They also include tests that we have added to thoroughly verify the hardware and help pinpoint the source of any problems we may find; stress testing the CPU, memory, disk and PCI bus in particular. Additionally, information obtained from the operating system, such as `/proc/meminfo` and `/proc/cpuinfo`, is checked for accuracy and consistency. Running all these tests could be a rather tedious process if done individually and interpreting their output may require more advanced system administration knowledge. To make this process easier, we have written a pair of wrapper scripts (`node_hw_test, node_hw_analyze`) to run each test with the correct parameters and parse the output. Not only do we check for pass/fail conditions, we compare benchmark results across the cluster to see if any nodes are performing sub-optimally or outside of comparable tolerances.

The interconnect tests include common tests such as Ping[22], Netperf[23] and Linpack, but we have found that some custom all-to-all communication tests and Myrinet hardware stress tests are often more useful in revealing and diagnosing problems related to network hardware. Installing the diagnostic module is as simple as running "`make diag`". Third party Makefiles are called when available and customized when necessary. Settings in the top level CIToolkit `Makefile` are leveraged to determine what architecture to compile the diagnostic tests for and where they should be installed.

5.7 Installation of the Run-time Environment

Since one of the goals of the CIToolkit was to separate run-time and infrastructure, a single startup point is provided for run-time initialization. After node initialization, a single startup script named "`rte`" (run-time environment), is executed which loads the appropriate run-time virtual machine based on the "`vmname`" attribute in the configuration database. This makes it easy for any run-time package to be initialized at the appropriate time; after the infrastructure is established. While we maintain a strict rule to not be dependent on any run-time system for the cluster infrastructure, many run-time systems can greatly benefit from the information available in the CIToolkit database and infrastructure.

6 Tools

The tools provided with the CIToolkit distribution range from very specific low-level tools to more general high-level tools intended for end-user interface. They are collectively called Layered Utilities. In this section we will describe a few examples from some of the categories of tools included:

- Database Tools
- Configuration Tools
- Low-Level Operation Tools
- High-Level Operation Tools (Csuite of Tools)
- Additional High-Level Tools, Interfaces, and Daemons

6.1 Database Tools

The database tools, also discussed in Section 5 (Installation Process), are tools that allow for various database operations like dumping and restoring the database and examining or modifying groups or device objects in the database. The dump and restore utilities are useful in initial database configuration to view what you have accomplished in an iterative manner. They are also useful in generating backups of the database, to be restored later if problems occur and the database is lost, or to transfer the configuration to another machine.

Grouping of device objects or other groups is a useful abstraction that simplifies management of large clusters. Within the configuration database, grouping can be done based on any scheme desired by the user. A method that we find useful is to create a group (or "collection") that contains node devices and their support devices, like terminal servers and external power supplies, which exist in the same rack. Another useful way to group devices is by associating them with the leader responsible for them. (The term leader as used here is a role that a node in the cluster can serve. This role is one of the many attributes that we find useful to define for devices.) This grouping can be leveraged by utilities in the toolkit to enhance scalability. These groups can be viewed, modified, or created using database tools.

Utilities are also provided to view, modify, or create device objects themselves. These utilities are useful in initial database generation, and after-the-fact modification of objects that have changed characteristics for any reason. For example, IP information exists as an attribute of a device object. If this information changes after initial database creation, available database tools can be utilized to modify this information. Discover, a utility previously mentioned in Section 5 (Installation Process) is also considered a database tool since it populates the database with information about the devices it discovers.

6.2 Configuration Tools

Configuration tools provide automatic generation of configuration files needed to initially build and/or maintain the completed cluster. Linux clusters use many files that fall into this category, for example:

- `/etc/hosts`
- `/etc/dhcpd.conf`
- `/etc/sysconfig/network-scripts/ifcfg-eth0`

These are just a few of the files that may be necessary to populate when installing and maintaining a cluster. The CIToolkit provides a tool to generate the `/etc/hosts` file for the entire cluster based on the cluster database. This utility can be used directly, but can also be leveraged by the utility that constructs the bootable hierarchy to create the hosts file and place it in the appropriate area. The `/etc/dhcpd.conf` configuration file can be very important for diskless clusters. A utility is also provided to create this file based on the cluster database information and is also leveraged by the utility that builds the bootable hierarchy. The `ifcfg-eth0` file is used to provide information on how to initialize network interface eth0, and is generated automatically by the utility used to build the diskless hierarchy. The information to generate this configuration file is obtained from the cluster database. If there are more interfaces, appropriate files are generated to initialize those interfaces. Since any important information about devices in the cluster is or can be stored in the cluster database, it becomes easy to add utilities when another configuration file is necessary to install or maintain the cluster.

6.3 Low-Level Operation Tools

Low-level operation tools are utilities provided in the CIToolkit that are orthogonal in their functionality, each one accomplishing a very specific task. Examples of low-level operations include:

- `power`
- `status`
- `boot`
- `console`

Each of the above is a task to accomplish as well as the actual command name. The low-level tools normally take a database object or list of objects as parameters. These objects can be devices or collections (groups) and the low-level tool will operate appropriately on the specified objects based on its purpose. For example, a user can boot a list of devices with a command like `boot node1 node2 node9` or a group that contains all the devices in a rack with a command like `boot rack3` (provided those devices and collection have been created in the configuration database). While each utility can be used alone, the low-level tools are also intended to be leveraged by the high-level operational tools of the CIToolkit.

6.4 High-Level Operation Tools (Csuite of Tools)

The high-level operation tools provide the user with a more abstract view of cluster maintenance. These tools frequently leverage low-level utilities to accomplish their functions. Many of these tools fall into what we term the csuite of utilities. Some of the utilities included in the csuite are:

- `cboot`
- `cstatus`
- `ccmd`

The command `cboot` allows the user to `power` (on, off, cycle) and `boot` the devices of a cluster. The `cstatus` command allows the user to determine the status (running, ready, unknown) of devices in the cluster. The `ccmd` command distributes specified commands to be executed in parallel on multiple node devices in the cluster. These three command line tools each utilize a combination of the lower-level tools to intelligently accomplish their tasks. They also provide a common interface to the user in the form of command line flags, which allow the user to easily execute complicated tasks.

As mentioned in the Section 3 (Goals), we have tried not to decide how a user will want to manage their cluster. Rather, we have attempted to provide utilities in our toolkit that allow the user as much flexibility as possible. All three command line tools allow for the following methods of specifying the device or devices on which to operate. The command accepts input from the user and decides the most efficient way to accomplish the operation. For large clusters this efficient distribution is essential to achieve the parallelism necessary to accomplish cluster maintenance tasks in a scalable and timely manner.

Each of the command line utilities in the csuite of commands allows for simple device or group specification. This allows the user to use these utilities just as they would the low-level utilities by simply providing a device or group name on the command line. For large clusters this method could result in either a lot of typing or not enough flexibility. The command line utilities also allow for simple flag specifications that represent device names and or groups. This provides a short cut approach that both shortens the command line and adds a large amount of flexibility. To illustrate this we will describe an example cluster and the naming scheme that represents the node devices in the cluster. It should be noted that the naming scheme is up to the site. We do not impose restrictions on how each site should name the devices in their cluster.

Our example cluster has a number of racks. The racks are named `r-#`, where `r` stands for rack and `#` is the rack number. There are 10 racks in our example cluster so the names of our racks will be `r-0` through `r-9`, inclusive. In each rack we have 32 nodes. A node name follows the designation `n-#` where `n` stands for node and `#` stands for the node number 0 through 31 inclusive. So the 1st node in rack 0 will have the designation `n-0.r-0`. The 32nd node in rack 10 will have the designation `n-31.r-9`. As a result the command line tools in the csuite of commands support flags like "`-r #`" to specify which rack number to operate on and "`-n #`" to specify which node to operate on. These flags can be used in a very free-form manner. Devices can be specified using many combinations of these flags. The following are some of the ways that the user can specify devices to operate on:

- A single node (`-n 0 -r 0`)
- List, or range of nodes in a single rack
  (`-n 0,2,4 -r 0`) or (`-n 0-5 -r 0`)
- List or range of nodes in a list or range of racks
  (`-n 0,2,4 -r 0,1,5`), (`-n 0-4 -r 5-9`)
- Just a list or range of racks (all nodes are included by default)
  (`-r 0,4,8`) or (`-r 5-9`)
- Combinations of lists and ranges of nodes or racks
  (`-n 0,2-5 -r 0`), (`-r 0,4-9`), (`-n 2 -r 3,5-7`)

Virtually any combination is supported which allows the user quite a bit of flexibility in specifying which devices to act upon.

Other methods that are supported for specifying the devices to act on are groupings based on attributes. The leader attribute mentioned previously can be leveraged to act on the leaders of the listed nodes when used in conjunction with the examples above. Another useful abstraction is to act on the cluster based on virtual machine specification. The virtual machine is a logical way to subdivide the cluster. (This abstraction is also very useful as a run-time concept as discussed previously.) For example, if all of the node devices in racks 0 through 5 are also members of a virtual machine called "default", we can act on all of these node devices with a simple command line flag/parameter combination of "`-vm default`". Providing just this flag to any one of the command line utilities in the csuite of commands allows the user to accomplish that functionality on the entire virtual machine. This capability can also be combined with the node and rack flags to form other groups of node devices to act on at the users discretion. For example, if both the virtual machine method and the node/rack method of device specification are used, the csuite of tools will take the intersection of to determine which devices to act upon. Here are some examples to illustrate this concept:

- Specify nodes that are in the virtual machine "default" $\cap$ nodes in rack 0
  (`-vm default -r 0`)
- Specify nodes 2-4 in racks 4-9 $\cap$ nodes in the virtual machine "default"
  (`-n 2-4 -r 4-9 -vm default`)

These are just two examples of ways to specify which nodes to act on when combining both the virtual machine method of specification with the rack/node method. It is left for the user to determine how to best manage their cluster. Specifying an individual device or group can also be incorporated in combination with one or both of these methods if desired. Other methods of specifying which devices to act on can easily be added to the capabilities of these commands

6.5 Additional High-Level Tools, Interfaces and Daemons

Each of the csuite of tools described above has the capability of performing its responsibilities by distributing work on its own, or interacting with an infrastructure of daemons running on nodes in the cluster. This daemon infrastructure was designed to add robustness and reliability especially for large clusters. It is in no way necessary for managing the cluster so sites that are very strict about consuming additional resources do not have to use them. The daemons are highly configurable and can be run on any combination of admin, leader and compute nodes. When available, the csuite of tools will contact these daemons, which will distribute the work as appropriate throughout the cluster, improving scalability and enhancing the capabilities of the existing tools. Along with this "Command Daemon" capability, the CIToolkit contains a "Status Daemon" which is a flexible monitoring process that gathers basic health information from cluster nodes, runs user selectable tests, and maintains a persistent storage of the results. These two daemons are designed to share as much code as possible, and can leverage each other, when appropriate, to perform their tasks most efficiently. At the time of this publication these daemon infrastructure components have been implemented and tested but have not stood the rigors of a production cluster. We don't feel implementations can be considered complete unless they have been proven in production to be stable and useful.

The CGUI is a graphical user interface that allows the less experienced user a more approachable way to interface with the cluster. Currently the toolkit provides a Graphical User Interface (GUI) implemented in Perl using the Perl/Tk[13] interface, as well as a Curses based GUI for managing the cluster from a terminal where an X window manager is not available or for remote administration over a slower SSH[24] connection. A web-based GUI is also under development to provide the user with whatever management tool is most appropriate for their site. These interfaces are implemented as another layer, leveraging lower-level utilities whenever possible. Each of these interfaces also leverage a common GUI abstraction layer, which encapsulates much of the GUI code, maximizes reuse, and makes it easy to add new capabilities to all three GUI's simultaneously.

Mentioned briefly in Section 5 (Installation Process), CHITS is a web-based hardware problem tracking tool which can be used both on site and remotely by system administrators, repair technicians, developers, etc. to automate the way in which hardware failures are reported, logged, repaired, verified, tracked and analyzed. The interface is streamlined for cluster systems by leveraging both the CIToolkit configuration database and the Device Class Hierarchy. CHITS provides more specific hardware issue tracking than generic help desk trouble ticket systems. This has been valuable to us in tracking both infant mortality issues and long-term trends in hardware failure. The system enables a more seamless process of communication from problem report to final resolution, which improves the turnaround time for repair. Minimizing component downtime has beneficial effects on overall system availability. CHITS provides easy access to repair history, which aids in the repair of similar failures, and can be used to generate spreadsheets and charts for hardware problem analysis. These reports can highlight trends in hardware failure that can be used for many purposes, such as preventive maintenance. Access to CHITS is controlled through the use of web server password authentication, either .htaccess, kerberos, or local password file based, and tools are provided in the *chits* module to manage the access controls.

A recent addition to the CIToolkit, `watch_console`, allows for greater capability and flexibility in monitoring console output of a single device, or multiple devices in parallel. While the low-level console utility is sufficient for individual device interaction, `watch_console` greatly expands on this capability by allowing multiple users to monitor the same device and buffering console output for post mortem viewing.

An increasing amount of effort has been directed at these high-level interfaces since the foundation of the CIToolkit has solidified. We anticipate not only improving the utilities mentioned above, but adding new utilities as opportunities for more efficient cluster management present themselves.

7 Future Work

In further pursuing the stated goals of the CIToolkit, we plan to develop additional tools in the areas of systems management, cluster usage data capture and presentation, and performance management. We plan to continue adding support for additional system architectures and new hardware. In addition, we will test new third-party software releases for capabilities that we have already included or are considering adding to the CIToolkit, and update our distribution accordingly. We plan to expand on the existing diagnostic tools, procedures and documentation to better perform the initial test and integration, continuing hardware validation, and "real-time" monitoring of a production cluster system.

As previously mentioned, libraries have been added to assist in the initial database creation and configuration, as well as subsequent manipulation by way of a command line interface using these libraries. Work to integrate this capability into any or all of the graphical user interfaces is under way, which will further ease interaction with the database.

Use of an LDAP database for the CIToolkit persistent object store has already been implemented and is in the beta stage. It will make managing mixed architecture clusters considerably easier and may improve scaling of clusters with a single database. We plan to test this more extensively and possibly make it the default database in our distribution.

While not yet required by our current installations, we plan to integrate the best available diskfull cluster management solution into the CIToolkit. This includes automating the installation of required third-party tools, leveraging the CIToolkit database for their configurations, and possibly augmenting their capabilities to make the tools more powerful.

While we have experienced great success thus far in building diskless clusters, which leverage NFS, we recognize that NFS has its limitations and is not optimal for very large clusters (in the 10,000 node range). Work is currently underway to explore a "lighter" compute node model which could be much more efficient in support of very large clusters used for scientific applications.

We plan to leverage the CIToolkit to provide manageability and configurability improvements to various run-time software environments, in particular Cplant. The intent is that this will provide add-on functionality to the run-time system for users that choose to also install the CIToolkit, making the cluster easier to configure and manage.

While we do not place requirements on the kernel, a module could be added to the CIToolkit to aid in the configuration of kernels for diskless clusters, various run-time systems, and high performance computing in general. Ideally, the configuration database will be folded back into the hardware integration and installation phase. Populating the configuration database should be part of the hardware specification process. With the database in place before the hardware and the addition of a new tools to the CIToolkit, it would be possible to have vendor-printed labels installed on all hardware and wiring diagrams generated to match the configuration database well ahead of time, thus speeding hardware installation and later troubleshooting.

8 Conclusions

Like many other sites, our first efforts in the area of cluster management and integration were not as organized or directed as the Cluster Integration Toolkit project has been. We learned, with some pain, that while not the most glamorous topic in the field of clusters, cluster management and integration is essential to the success of any serious cluster project. This necessitates the same rigorous approach applied to other topics in cluster computing be applied here, with equal vigor. Our investments in the development of goals, design, and adherence to the principles therein, have paid off in the resulting product. Our goals have been achieved, and as the project continues, our time can be spent on improvements and supplemental capabilities rather than re-writing the same code for every new cluster design or hardware technology that comes along.

Focusing on the similarities of clusters as a whole rather than the uniqueness of individual installations has paid off with a flexible product which has ported easily to new clusters at our site, and has proven useful at other installations. Throughout the evolution of this product, we have reminded ourselves that we should not make assumptions about how others would like to manage their cluster. We have frequently been the beneficiaries of this and have been glad we kept our resolve.

9 Acknowledgements

We would like to acknowledge the contributions of the Scalable Systems Integration, Scalable Computing Systems, and Scientific Computing organizations at Sandia National Labs. The CIToolkit project, started as part of the larger Cplant effort, has involved many personnel from these and other departments.

Their contributions have been significant to the success of this project and their time and efforts are greatly appreciated. High Performance Technologies Inc. has played an ever-increasing role in the implementation of the CIToolkit, the skill and professionalism of the technologists and management has been invaluable. We would also like to recognize the contributions of Doug Doerfler[1] and Neil Pundit[1] who have consistently supported our efforts and allocated the necessary resources without hesitation.

References

1. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.
{jhlaros,lee,galagun,mrepper,jrstear,dwdoerf,pundit}@sandia.gov
2. High Performance Technologies Incorporated, 11955 Freedom Drive, Suite 1100 Reston, VA 20190 {ndauchy,jvasak}@hpti.com
3. Compaq Federal LLC, 5130 Masthead St NE Suite B., ABQ NM 87109 rklundt@sandia.gov
4. *"An Extensible, Portable, Scalable Cluster Management Software Architecture"*, Laros, Ward, Dauchy, Brightwell, Hudson, Klundt, published in the proceedings of the IEEE International Conference on Cluster Computing 2002.
5. Linux Trademark held by Linus Torvalds
6. Cplant is a registered trademark of Sandia National Labs. R.B. Brightwell, L. A. Fist, D. S. Greenberg, T. B. Hudson, M. J. Levenhagen, A. B. Maccabe and R. E. Riesen, *"Massively Parallel Computing Using Commodity Components"*.
7. LinuxBios, http://www.linuxbios.org
8. Perl, http://www.perl.org
9. GNU Database Manager, http://www.gnu.org/software/gdbm/gdbm.html
10. Lightweight Directory Access Protocol, http://www.openldap.org/
11. Microsoft Corporation, One Microsoft Way, Redmond, Washington 98052-6399 http://www.microsoft.com
12. Sharp Electronics Corp, http://www.sharpusa.com
13. Comprehensive Perl Archive Network, http://www.cpan.org
14. Myricom Inc. 325 N. Santa Anita Ave. Arcadia, CA 91006, Myrinet and Myricom are registered trademarks of Myricom Inc. http://www.myri.com
15. Implementing Scalable Diskless Cluster Systems using the Network File System (NFS), Laros and Ward, (Not yet published)
16. Cfengine - a configuration engine, http://www.cfengine.org
17. Stream - http://www.cs.virginia.edu/stream/
18. Memtest - http://www.qcc.ca/ charlesc/software/memtester/index.shtml
19. Linpack - http://www.netlib.org/benchmark/hpl/
20. hdparm - Author Mark Lord, reference man page on most UNIX systems
21. NAS benchmarks - http://www.nas.nasa.gov/Software/NPB/
22. ping is part of the iputils package distributed with Linux distributions
23. Netperf - http://www.netperf.org/netperf/NetperfPage.html
24. open ssh, http://www.openssh.org/